NOTACIÓN DE BACKUS-NAUR (BNF)

Backus–Naur form (BNF)

Autor 1: Diego Alejandro Castro Cardona

*Ingeniería de Sistemas, Universidad Tecnológica de Pereira*

Correo-e: d.castro4@utp.edu.co

***Resumen*— Es un metalenguaje de programación que se ha constituido a lo largo del tiempo en uno de los sistemas de notación técnica más frecuentemente usados en computación. Tal como se asegura en una de sus páginas web, la fuente y creación del mismo, surgió a partir de las teorías lingüísticas de Noam Chomsky. Peter Naur tomó el BNF para el desarrollo de las gramáticas de los lenguajes de programación, pero también para representar en un lenguaje formalizado partes de las estructuras gramáticas de la lengua natural. Primeramente, John Backus y posteriormente Peter Naur establecieron una sintaxis que responde a la lógica matemática y a una estructura algorítmica. El objetivo fue que esa gramática les permitiera crear una base de instrucción para describir y modelar distintos tipos de textos, a saber: documentos que estén bajo formato, conjunto de instrucciones de programación, protocolos de comunicación, lenguajes de programación, notación para las gramáticas y sintaxis de los lenguajes de programación de la computadora, etc. Por supuesto como todo metalenguaje.**

***Palabras clave—* Tecnología, Informática, Tics, Computación, Backus Naur, Assembler, Metalenguaje. Lenguajes de programación.**

***Abstract*— It is a programming metalanguage that has been constituted over time in one of the most frequently used systems of technical notation in computing. As stated in one of its web pages, the source and creation of it, arose from the linguistic theories of Noam Chomsky. Peter Naur took the BNF for the development of the grammar of programming languages, but also to represent parts of the grammar structures of the natural language in a formalized language. First, John Backus and later Peter Naur established a syntax that responds to mathematical logic and an algorithmic structure. The objective was that this grammar would allow them to create an instruction base to describe and model different types of texts, namely: documents that are under format, set of programming instructions, communication protocols, programming languages, notation for grammars and syntax of computer programming languages, etc. Of course like all metalanguage.**

***Key Word* —Technology, Computing, Induction, Backus Naur Form, programming languages.**

1. INTRODUCCIÓN

Este documento es una guía de formato o plantilla. Puede obtenerse una copia de la página del curso, o incluso puede buscar por otras versiones semejantes en internet. La idea de esta sección, es dar una introducción al tema que se tratará en el artículo, de forma concisa y que permita al lector prepararse para los contenidos siguientes.

La inducción es el proceso en el cual las personas principiantes en un tema tienen una explicación para saber de qué se trata.

1. CONTENIDO

**Teoría Simple:**

La notación BNF fue creada por J. Backus y P.Naur para describir la sintaxis del lenguaje ALGOL 60. Utiliza reglas que se construyen con tres tipos de símbolos metasímbolos : son los propios de la BNF. símbolos terminales : son los que se usan en el texto del programa. símbolos no terminales : son los demás. Se definen usando combinaciones de símbolos terminales, no terminales y metasímbolos

Metasímbolos:

***::=*** *se define como*

***|*** *ó*

***{ }*** *repetición*

***[ ]*** *opcional*

*::= Se define como*

*| Opción alternativa*

*( a | b ) a ó b, pero no ambos*

*[ algo ] 0 ó 1 vez algo*

*{ algo } 0 ó más veces algo*

*'xyz' símbolo terminal*

*. final de la regla*

Una especificación de BNF es un sistema de reglas de derivación, escrito como:

*<****simbolo****> ::= <expresión con símbolos>*

donde <símbolo> es un no terminal, y la expresión consiste en secuencias de símbolos o secuencias separadas por la barra vertical, '|', indicando una opción, el conjunto es una posible substitución para el símbolo a la izquierda. Los símbolos que nunca aparecen en un lado izquierdo son terminales.

**Ejemplos:**

Gramática para representar números con decimales

*numero ::= entPos*

*| entPos '.' entPos*

*entPos ::= digito*

*| digito entPos*

*digito ::= '0' | '1' | '2' | '3' | '4'*

*| '5' | '6' | '7' | '8' | '9'*

*numero ::= digito + ( '.' digito + ) ?*

*digito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'*

Para definir un dígito (numero) hacen falta 10 reglas:

*digito ::= '0'.*

*digito ::= '1'.*

*digito ::= '2'.*

*digito ::= '3'.*

*digito ::= '4'.*

*digito ::= '5'.*

*digito ::= '6'.*

*digito ::= '7'.*

*digito ::= '8'.*

*digito ::= '9'.*

También se puede abreviar con una regla con alternativas:

*digito ::= '0' | '1' | '2' | . . . | '9'.*

Esta última regla se leería en español como 'un digito se define como un cero o un uno o un dos ... o un nueve.

La definición de un signo:

*signo ::= '+' | '-' .*

La definición de un entero sin signo

entero\_sin\_signo ::= digito | digito entero\_sin\_signo.

Otro caso de ejemplo podría ser la escritura de caracteres, tales como se ven acontinuacion:

*<letraIng> ::= “A” | “B” | “C” | “D”| “E” | “F” | “G” | “H” | “I” | “J”| “K” | “L” | “M” | “N” | “O” | “P”| “Q” | “R” | “S” | “T” | “U” | “V”| “W” | “X” | “Y” | “Z” | “a” | “b”| “c” | “d” | “e” | “f” | “g” | “h”| “i” | “j” | “k” | “l” | “m” | “n”| “o” | “p” | “q” | “r” | “s” | “t”| “u” | “v” | “w” | “x” | “y” | “z”*

Ejemplo en BNF, la clásica expresión gramática seria:

*<expr> ::= <term> "+" <expr>*

*| <term>*

*<term> ::= <factor> "\*" <term>*

*| <factor>*

*<factor> ::= "(" <expr> ")"*

*| <const>*

*<const> ::= integer*

Naturalmente, podemos definir gramaticas para reglas en BNF:

*rule → name ::= expansion*

*name → < identifier >*

*expansion → expansion expansion*

*expansion → expansion | expansion*

*expansion → name*

*expansion → terminal*

Otro ejemplo es hacer el formato de fecha y hora tomado de RFC 5322:

*date-time = [ day-of-week "," ] date time [CFWS]*

*day-of-week = ([FWS] day-name) / obs-day-of-week*

*day-name = "Mon" / "Tue" / "Wed" / "Thu" /*

*"Fri" / "Sat" / "Sun"*

*date = day month year*

*day = ([FWS] 1\*2DIGIT FWS) / obs-day*

*month = "Jan" / "Feb" / "Mar" / "Apr" /*

*"May" / "Jun" / "Jul" / "Aug" /*

*"Sep" / "Oct" / "Nov" / "Dec"*

*year = (FWS 4\*DIGIT FWS) / obs-year*

*time = time-of-day zone*

*time-of-day = hour ":" minute [ ":" second ]*

*hour = 2DIGIT / obs-hour*

*minute = 2DIGIT / obs-minute*

*second = 2DIGIT / obs-second*

*zone = (FWS ( "+" / "-" ) 4DIGIT) / obs-zone*

Las expresiones regulares se ubican justo debajo de las gramáticas libres de contexto en poder descriptivo: podría reescribir cualquier expresión regular en una gramática que represente los fragmentos coincidentes con la expresión. Pero lo contrario no es cierto: no toda gramática se puede convertir en una expresión regular equivalente.

Para ir más allá del poder expresivo de las gramáticas libres de contexto, uno necesita permitir un grado de sensibilidad al contexto en la gramática.

La sensibilidad al contexto significa que los símbolos terminales también pueden aparecer en los lados izquierdos de las reglas.

**Biografías**

**John Backus:**

John Backus (Filadelfia, 3 de diciembre de 1924 - Oregón, 17 de marzo de 2007)

fue un científico de la computación estadounidense.

Ganador del Premio Turing en 1977 por sus trabajos en sistemas de programación de alto nivel, en especial por su trabajo con FORTRAN.

Para evitar las dificultades de programación de las calculadoras de su época, en 1954 Backus se encargó de la dirección de un proyecto de investigación en IBM para el proyecto y realización de un lenguaje de programación más cercano a la notación matemática normal. De ese proyecto surgió el lenguaje FORTRAN, el primero de los lenguajes de programación de alto nivel que tuvo un gran impacto, incluso comercial, en la emergente comunidad informática.

Tras la realización de FORTRAN, Backus fue un miembro muy activo del comité internacional que se encargó del proyecto de lenguaje ALGOL. En ese contexto propuso una notación para la representación de las gramáticas usadas en la definición de un lenguaje de programación (las llamadas gramáticas libres de contexto). Tal notación se conoce como Notación de Backus-Naur (Backus-Naur Form o BNF) y une al nombre de Backus al de Peter Naur, un informático europeo del comité ALGOL que contribuyó a su definición.

En los años 1970, Backus se interesó sobre todo por la Programación funcional, y proyectó el lenguaje de programación FP, descrito en el texto que le sirvió para ganar el premio Turing, "Can Programming be Liberated from the Von Neumann Style?" Se trata de un lenguaje de uso fundamentalmente académico, que sin embargo animó un gran número de investigaciones. El proyecto FP, transformado en FL, se terminó cuando Backus se jubiló en IBM, en 1991.

John Backus falleció el sábado 17 de marzo de 2007, a la edad de 82 años en su casa en Ashland, Oregón por causas naturales, de acuerdo a la declaración de su familia.

**Peter Naur:**

Peter Naur (nacido el 25 de octubre de 1928), es un científico danés pionero en informática y ganador del Premio Turing en 2005.

Biografía

La N de la notación BNF, usada en la descripción de la sintaxis de la mayoría de los lenguajes de programación, se usa en alusión a su apellido. Naur contribuyó en la creación del lenguaje de programación Algol 60. Empezó su carrera como un astrónomo, pero su encuentro con las computadoras lo hizo cambiar de carrera. A Naur no le agrada el concepto Ciencia de la computación ("Computer Science" en inglés) y sugiere llamarlo Datalogía ("Datalogy" en inglés). Este término fue adoptado en la mayoría de los países escandinavos (Dinamarca y Suecia).

Trabajó en el Regnecentralen (instituto de computación danés), en el Instituto Niels Bohr y en la Universidad Técnica de Dinamarca. De 1968 a 1998 trabajó como profesor en la Universidad de Copenhague.

Es conocido por su crítica al uso de los métodos formales en programación. Así mismo, basado en su inclinación desde el empirismo, critica el uso que le dan los filósofos a la lógica para describir la ciencia. Critica igualmente a psicólogos que todavía se basan en teorías del conductismo y el constructivismo.

En los últimos años ha estado desarrollando una teoría del pensamiento humano que ha denominado Teoría Sinapsis-Estado de Vida Mental ("The Synapse-State Theory of Mental Life" en inglés).

**Noam Chomsky:**

(Noam Abraham Chomsky; Filadelfia, 1928) Lingüista y filósofo estadounidense. Fue introducido en la lingüística por su padre, especializado en lingüística histórica del hebreo. Estudió en la Universidad de Pensilvania, donde se doctoró en 1955 con una tesis sobre el análisis transformacional, elaborada a partir de las teorías de Z. Harris, de quien fue discípulo. Entró entonces a formar parte como docente del Massachusetts Institute of Technology, del que fue profesor desde 1961.

Es autor de una aportación fundamental a la lingüística moderna, con la formulación teórica y el desarrollo del concepto de gramática transformacional, o generativa, cuya principal novedad radica en la distinción de dos niveles diferentes en el análisis de las oraciones: por un lado, la «estructura profunda», conjunto de reglas de gran generalidad a partir de las cuales se «genera», mediante una serie de reglas de transformación, la «estructura superficial» de la frase.

1. CONCLUSIONES

Para concluir, la notación Backus naur nos permite desde expresar gramáticas libres de contexto, es decir que nos permite crear una manera mas ordenada y formal de escribir lenguajes formales, esta idea es de transcribir estructuras de lenguajes con reglas de reescritura, además todo este proceso se hace desde las formas básicas, desde cero, BNF no es más que el lenguaje de lenguajes.

Asi, nosotros podemos crear cualquier tipo de lenguaje desde sus inicios utilizando esta técnica y los diferentes recursos del Backus Naur Form.

**Impacto:**

Es el ejemplo más conocido de un metalenguaje, es decir, uno que describe sintácticamente un lenguaje de programación. Usando BNF es posible especificar qué secuencias de símbolos constituyen un programa sintácticamente válido en un idioma dado.

Su impacto en relación a los lenguajes de programación:

Durante el período inmediatamente posterior a la publicación del informe ALGOL 60, BNF fue la base de muchos sistemas compilador-compilador. Algunos utilizaron directamente BNF como "Un compilador dirigido de sintaxis para ALGOL 60" desarrollado por Edgar T. Irons y "Un sistema de compilación de compiladores" desarrollado por Brooker y Morris. Otros lo cambiaron a un lenguaje de programación. Los metacompiladores de Schorre lo convirtieron en un lenguaje de programación con solo unos pocos cambios. <nombre de clase> se convirtió en identificadores de símbolos, eliminando el <,> y utilizando cadenas entre comillas para los símbolos del idioma de destino. La agrupación aritmética proporcionó una simplificación que se eliminó usando clases donde la agrupación era su único valor. La regla de expresión aritmética META II muestra el uso de agrupación. Las expresiones de salida colocadas en una regla META II se usan para generar código y etiquetas en un lenguaje ensamblador. Las reglas en META II son equivalentes a las definiciones de una clase en BNF. La utilidad yacc de Unix se basa en BNF con una producción de código similar a META II. yacc se usa más comúnmente como un generador de analizadores sintácticos, y sus raíces son obviamente BNF. BNF hoy es uno de los lenguajes informáticos más antiguos que aún se utilizan.

REFERENCIAS

1. <https://es.wikipedia.org/wiki/Razonamiento_inductivo>
2. <https://translate.google.com/?hl=es>
3. <https://www.researchgate.net/publication/325881080_Backus-Naur_Form_reglas_generativas_y_descripcion_de_un_meta-lenguaje>
4. <https://www.fing.edu.uy/inco/cursos/teoleng/obligatorio/presentacion.pdf>